

# PHP – An Introduction for C++ Programmers

© 2009 by Dr. Frank McCown - Harding University  
Last updated: March 20, 2009

## Overview

The following is a summary of the various aspects of the PHP language for beginners who have a C++ background. It is in no way intended to be a complete or comprehensive examination of PHP. See the official PHP manual for more detailed information: <http://www.php.net/manual/en/>. All syntax contained in this guide are for PHP 5 and may not be compatible with previous versions of PHP.

## Background

- Nerdy recursive acronym: PHP: Hypertext Preprocessor (originally named Personal Home Page Tools)
- Invented by Rasmus Lerdorf in 1994 and is now under the Apache Software Foundation. Licensed under the GPL and is free. Current version as of April 2008 is PHP 5.2.5.
- Popular server-side technology for Apache web servers. Competing technologies are CGI, Microsoft's ASP.NET, Macromedia's ColdFusion, and Sun's Java Server Pages.
- Available on a variety of web servers (Apache, IIS, Netscape, etc.) and operating systems (Windows, Linux, UNIX, Mac OS X, etc.).
- Supports many types of databases – MySQL, Oracle, ODBC (for MS Access and SQL Server), etc.
- Scripting language – gets interpreted instead of being compiled. Executed on the web server, not the client.
- Syntax is very similar to Perl and C. Variables are case sensitive, function names are not, and statements must be terminated with a semicolon.
- Support for object-oriented programming (OOP).

## On-line Resources

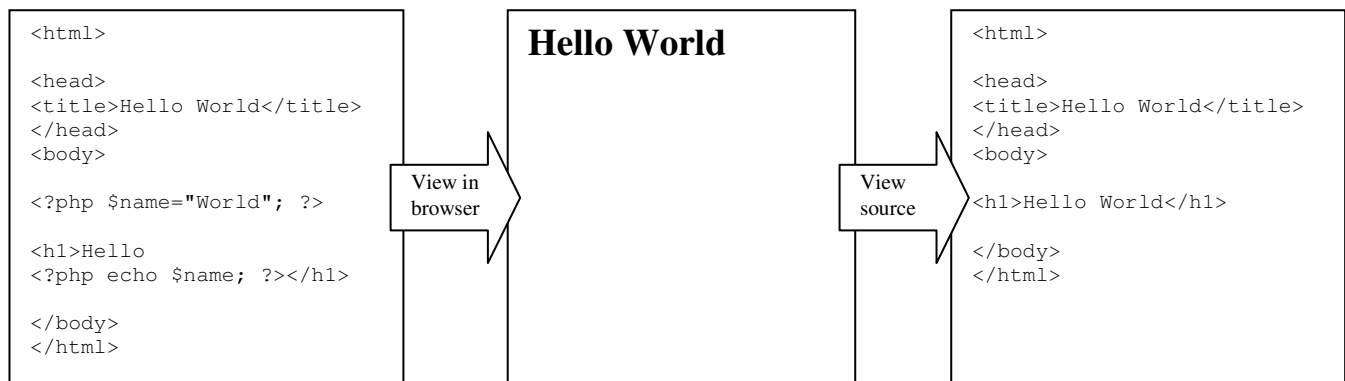
[www.php.net](http://www.php.net) – PHP distribution, tutorials, newsgroups, and more.

[www.phpfreaks.com](http://www.phpfreaks.com) - PHP and MySQL tutorials, scripts, forums, and more.

[www.phpbuilder.com](http://www.phpbuilder.com) – Collection of PHP resources.

## Hello World

If your web server supports PHP, type this example into a text file called hello.php and access it in your browser by typing the complete URL (e.g., <http://www.example.com/hello.php>). Your .php file will need word read permissions.



## Table of Contents

I.	Inserting code into HTML .....	2
II.	Comments .....	2
III.	Variables and Data Types .....	2
IV.	Operators.....	3
V.	Input/Output.....	3
VI.	Control Structures.....	4
VII.	Arrays .....	5
VIII.	Functions .....	6
IX.	Strings.....	7
X.	Regular Expressions .....	8
XI.	Exception Handling.....	9
XII.	File I/O .....	9
XIII.	Including Scripts within Scripts .....	10
XIV.	Web Input and Sessions.....	11
XV.	Classes and Objects.....	12
XVI.	Database Connectivity.....	13

### I. Inserting code into HTML

- A. PHP code should be placed between `<? code ?>` or `<?php code ?>` tags. The second method is preferred so your scripts are XML compatible.

### II. Comments

All three style are legal:

```
# Perl style single line comment
// Single line comment
/* Multiple
   line comments */
```

### III. Variables and Data Types

- A. Always starts with `$` and letter or underscore. Can be composed of numbers, underscores, and letters.

```
$my_var = 10;
$a_2nd_var = "bison";
```

- B. Data types: integers, doubles (numbers with a decimal point), boolean (true or false), NULL, strings, arrays, objects, and resources (like database connections). Variables do not have to be declared and neither do their data types.
- C. If you try to use a variable before setting it to a value, the default error-reporting setting will give you an "Undefined variable" warning.
- D. All variables have local scope (i.e., they are accessible only within the function or block in which they are initialized). Global variables may only be accessed within a function by using the `global` keyword.

```
$x = "test";

function display() {
    global $x;
    echo $x;
}
```

- E. Constants are defined using `define` and by convention are usually named in ALL CAPITALS.

```
define("PI", 3.14);
define("HEADING", "<h1>My Web Site</h1>");
$area = PI * $radius * $radius;
print(HEADING);
```

## IV. Operators

### A. Assignment

1. `=` `+=` `-=` `/=` `*=` `%=` `++` `--` - just like C++.
2. `.=` - string concatenation operator (see strings section).

### B. Arithmetic

1. `+` `-` `*` `/` `%` - just like C++.

### C. Relational

1. `==` `!=` `<` `>` `<=` `>=` - just like C++
2. `===` - true if arguments are equal and the same data type.
3. `!==` - true if arguments are not equal or they are not of the same data type.

### D. Logical

1. `&&` `||` `!` - essentially like C++
2. `and` `or` - like `&&` and `||` but have lower precedence than `&&` and `||`.
3. `xor` - true if either (but not both) of its arguments are true.

## V. Input/Output

- A. `print` and `echo` are used to print to the browser.

```
echo "Go Bisons";
echo("Go Bisons"); // same thing
print("Go Bisons"); // same thing
```

- B. `print` can only accept one argument, and `echo` can accept any number of arguments. `print` returns a value that indicates if the print statement succeeded.

C. Variables are interpolated inside of strings unless single quotes are used.

```
$a = "guts";  
echo "You have $a."; // prints "You have guts."  
echo 'You have $a.'; // prints "You have $a."
```

D. Escape sequences: \n (newline), \r (carriage-return), \t (tab), \\$ (\$), \" ("), \\ (\)

```
echo "a\\tb\$c"; // prints "a\ b$c"  
echo 'a\\tb\$c'; // prints "a\\tb\$c"
```

E. printf works like C's counter-part.

```
$title = "X-Men";  
$amount = 54.235;  
printf("The movie <b>%s</b> made %2.2f million.", $title, $amount);  
// prints "The movie <b>X-Men</b> made 54.23 million."
```

F. PHP typically does not run from the command-line, but input from the keyboard can be accessed using the fopen function with "php://stdin". See the file I/O section for more information.

G. Output shortcut from within HTML:

```
Your name is <b><?=$name ?></b>.
```

is the same as

```
Your name is <b><?php echo $name ?></b>.
```

## VI. Control Structures

A. Choice structures

1. **if** (\$x > 0)  
    \$y = 5; // {} not required for only one statement
2. **if** (\$a) { // tests if \$a is true or non-zero or a non-empty string  
    print(\$b);  
    \$b++;  
}  
    **else**  
    print(\$c);
3. **if** (\$a > \$b)  
    print "a is bigger than b";  
    **elseif** (\$a == \$b) // use "elseif" or "else if"  
    print "a is equal to b";  
    **else**  
    print "a is smaller than b";
4. **switch** (\$vehicle\_type) { // works for integers, floats, or strings  
    **case** "car": \$car++; break;  
    **case** "truck": \$truck++; break;  
    **case** "suv": \$suv++; break;  
    **default**: \$other++;  
}

## B. Looping structures

1. **while** ( $\$n < 10$ ) {  
    print(" $\$n$  ");  
     $\$n++$ ;  
}
2. **do** {  
    print(" $\$n$  ");  
     $\$n++$ ;  
} **while** ( $\$n < 10$ );
3. **for** ( $\$n = 1$ ;  $\$n < 10$ ;  $\$n++$ )  
    print(" $\$n$  ");
4. **foreach** ( $\$myarray$  **as**  $\$item$ )  
    print(" $\$item$  ");

## VII. Arrays

A. Summary of all array functions in the PHP core: <http://www.php.net/manual/en/ref.array.php>

B. Arrays can have any size and contain any type of value. No danger of going beyond array bounds.

```
 $\$my\_array[0] = 25$ ;  
 $\$my\_array[1] = "Bisons"$ ;
```

C. PHP arrays are **associative arrays** which allow element values to be stored in relation to a key value rather than a strict linear index order.

```
 $\$capitals["CO"] = "Denver"$ ;  
 $\$capitals["AR"] = "Little Rock"$ ;
```

D. Initialize an array:

```
 $\$colors = array("red", "green", "blue")$ ;  
print("The 2nd color is  $\$colors[1]$ ."); // prints green  
  
 $\$capitals = array("CO" => "Denver", "AR" => "Little Rock")$ ;  
print(" $\$capitals[CO]$ "); // prints Denver, no quotes around key inside ""
```

E. Print contents of an array:

<pre><code>print_r(<math>\\$colors</math>);</code></pre>	<pre><code>print_r(<math>\\$capitals</math>);</code></pre>
produces:	produces:
<pre><code>Array (     [0] =&gt; red     [1] =&gt; green     [2] =&gt; blue )</code></pre>	<pre><code>Array (     [CO] =&gt; Denver     [AR] =&gt; Little Rock )</code></pre>

F. Pull values out of an array:

```
 $\$colors = array("red", "green", "blue")$ ;  
list( $\$c1$ ,  $\$c2$ ) =  $\$colors$ ;  
print(" $\$c1$  and  $\$c2$ "); // prints "red and green"
```

G. Delete from an array:

```
unset( $\$colors[1]$ ); //  $\$colors$  now contains red and blue at indexes 0 and 2.
```

#### H. Extracting array keys and values:

```
$states = array_keys($capitals); // $states is ("CO", "AR")
$cities = array_values($capitals); // $cities is ("Denver", "Little Rock")
```

#### I. Iterating through an array:

```
$heroes = array('Spider-Man', 'Hulk', 'Wolverine');
foreach ($heroes as $name)
    print("<br>$name"); // prints all three in order

foreach ($capitals as $state => $city)
    print("<br>$city is the capital of $state.");
```

#### J. Treat like a stack:

```
array_push($heroes, 'Iron Man'); // Pushed onto end of array
$h = array_pop($heroes); // Pops off last element (Iron Man)
```

#### K. Size of an array:

```
$num_items = count($heroes); // returns 3
```

#### L. Sort an array:

```
sort($heroes); // Heroes are now in alphabetical order (lowest to highest)
rsort($heroes); // Reverse alphabetical order (highest to lowest)
```

## VIII. Functions

- A. PHP pre-defined functions are documented at <http://www.php.net/manual/en/funcref.php>.
- B. Functions may be declared anywhere in the source code (i.e., they do not need to be defined before they are called like C++).
- C. Function names are case-insensitive, though it is usually good form to call functions as they appear in their declaration.
- D. Defining and calling

##### 1. General form:

```
function func_name($param_1, $param_2, ..., $param_n) {
    // code

    return $retval; // optional: can return a scalar or an array
}
```

2. Call: `$result = func_name($arg1, $arg2, ..., $argn);`

#### E. Parameter passing and returning values

- 1. Arguments may be passed by value (default) or by reference (using &). Default argument values can also be used which must be initialized in the parameter list. Variable-length argument lists are also supported but are not covered here.

```

// Pass by value
function sum($a, $b) {
    return $a + $b;
}

// Pass by reference
function swap(&$a, &$b) {
    $temp = $a;
    $a = $b;
    $b = $temp;
}

// Default arguments must be on right side
function say_greeting($name, $greeting="Hello") {
    print "$greeting, $name!";
}

say_greeting("Susan"); // Hello, Susan!
say_greeting("Rita", "Hola"); // Hola, Rita!

```

## 2. Passing an array by value and by reference:

```

// Pass by value
function sum_array($values) {
    $sum = 0;
    foreach ($values as $num)
        $sum += $num;
    return $sum;
}

$nums = array(1, 2, 3);
print "Sum of array = " .
    sum_array($nums); // 6

// Pass by reference
function randomize(&$nums) {
    for ($i = 0; $i < 10; $i++)
        $nums[$i] = rand(0, 100); // 0-100
}

$n = array();
randomize($n); // Place 10 random nums in $n

```

## 3. Return an array:

```

// Return an array
function special_nums() {
    return array(3.142, 2.718, 1.618);
}

list($pi, $euler, $phi) = special_nums();

```

# IX. Strings

## A. Concatenation

```
$full_name = $first_name . " " . $last_name; // results in "Bob Smith"
```

## B. PHP string functions. View the complete list at <http://www.php.net/manual/en/ref.strings.php> .

int **strlen**(\$str) Returns string length.

int **strcmp**(\$str1, \$str2)

Returns < 0 if str1 is less than str2; > 0 if str1 is greater than str2, and 0 if they are equal. (strcasecmp for case-insensitive comparison.) The < > == operators can also be used if both arguments are strings. strcmp is useful if an argument may not be a string and has to be converted into one.

string **strstr**(\$text, \$search)

Returns first occurrence of \$search in \$text, FALSE if not found. (stristr for case-insensitive search.)

string **str\_replace**(\$find, \$replace, \$text)

Replaces all occurrences of \$find with \$replace in \$text.

string **chop**(\$str) Removes all white space at end of string.

string **ltrim**(\$str) Removes all white space at beginning of string.

string **trim**(\$str)    Removes all white space at beginning and end of string.

## X. Regular Expressions

- A. Regular expressions are patterns that can be used to match text strings. They can be used, for example, to determine if a string is a legal email address or phone number. PHP regular expressions are implemented very similarly in other programming languages. For a complete reference, see <http://us2.php.net/manual/en/ref.pcre.php>.
- B. The examples here use Perl regular expressions which require forward slashes ("/) around the pattern.
- C. Matching patterns

- 1. Find the given pattern anywhere in the string

```
if (preg_match("/ard/", "Harding"))
    echo "Matches";
else
    echo "No match";
```

- 2. Special symbols

<code>\d</code>	any digit (0-9)	<code>{3}</code>	match only three of these
<code>\s</code>	any white space (space, tab, EOL)	<code>?</code>	match zero or one character
<code>\w</code>	any word char (a-z, A-Z, 0-9, _)	<code>*</code>	match zero or more characters
<code>.</code>	any character except EOL	<code>+</code>	match one or more characters
<code>[abc]</code>	a, b, or c	<code>^abc</code>	match at the beginning of the string
<code>[^a-z]</code>	not any char between a and z	<code>abc\$</code>	match at the end of the string

- 3. Email address example

```
$email = 'first_name.last_name@domain.Com';
$regexp = "/^[\\w.]+@[\\w.]+[a-z]{2,4}$/i";    // i switch for case-insensitive match
if (preg_match($regexp, $email))
    echo "Match email";
```

- 4. Remembering matched patterns

```
if (preg_match('/(\\d\\d):(\\d\\d) (am|pm)/', '03:15 pm', $matches)) {
    echo "Hour: $matches[1]\\n";    // 03
    echo "Min: $matches[2]\\n";    // 15
    echo "Ending: $matches[3]\\n";    // pm
}
```

- 5. Match all occurrences

```
preg_match_all('/.ar/', 'the car was far from the bar', $matches);
print_r($matches[0]);    // Prints car, far, bar
```

- D. Replacing patterns

- 1. Simple replacement

```
$new = preg_replace('/hard/', 'easy', 'this is hard!');    // Returns "this is easy!"
```

## 2. Replacement with remembered matches

```
// Convert MM/DD/YYYY to YYYY-MM-DD (must escape / in regex)
$date = preg_replace("/(\\d\\d)/(\\d\\d)/(\\d\\d\\d\\d)/", "$3-$1-$2", "08/15/2008");
```

## E. Array processing

### 1. Split string into an array

```
$names = 'Fred, Erin, Alex, Sunshine';
$names_array = preg_split('/[, ]+/', $names); // Returns one name in each slot
```

### 2. Find all items in an array that match a regex

```
// Returns only Erin and Alex
$startsWith_vowel = preg_grep('/^[aeiou]/i', $names_array);
```

## XI. Exception Handling

### A. PHP uses exception (error) handling much like C++.

```
function divide($x, $y) {
    if ($y == 0)
        throw new Exception('Division by zero.');
```

```
    else
        return $x / $y;
}

try {
    echo divide(5, 2) . "\n"; // Prints 2.5
    echo divide(4, 0) . "\n"; // Causes exception to be thrown
}
catch (Exception $e) {
    echo 'Caught exception: ', $e->getMessage(), "\n";
}
```

### B. Exceptions that are not caught cause the script to fail with a fatal error.

### C. Other information about the exception like the line number and the file name in which the exception occurred is also available. See <http://us.php.net/manual/en/language.exceptions.php>.

## XII. File I/O

### A. PHP can access any file that is stored on the web server, as long as it has the proper permissions.

### B. HTTP, FTP, and STD read/write can also be used with file functions.

### C. See <http://www.php.net/manual/en/ref.filesystem.php> for functions that set file permissions, copy and delete files, access file modification times, and a lot more.

### D. Open file with `fopen`, close with `fclose`. File open modes:

1. “r” – Read from existing file.
2. “r+” – Read and write to already existing file.
3. “w” – Write to file, creating the file if it doesn’t already exist, erasing it if it does.
4. “w+” – Write and read, creating the file if it doesn’t already exist, erasing it if it does.
5. “a” – Append to end of file whether it exists or not.

6. "a+" – Append to end of file, doubling file contents if you read the file in as a string, edit it, and write it back to the file.

E. Reading from a file. File must have proper read permissions. If file is not owned by "nobody", it must have world read permissions.

1. Read entire file:

```
$fd = fopen("myfile.txt", "r") or die("Can't open myfile.txt for reading.");
$entire_file = fread($fd, filesize("myfile.txt"));
print $entire_file;
fclose($fd);
```

\*The `die` operator is useful for halting executing of a PHP script and giving an error message. You may also call `exit` which does the same thing.

2. Read line by line:

```
while (!feof($fd)) {
    $line = fgets($fd, 4096); // 4096 is the max bytes per line
    print $line . "<br>";
}
```

F. Writing to a file. The directory containing the file to be written must have at least world execute and write permissions. Default owner of a created file will be "nobody".

```
$fd = fopen("myfile.txt", "w") or die("Can't write to myfile.txt.");
fwrite($fd, "This is output.");
fclose($fd);
```

### XIII. Including Scripts within Scripts

A. A file containing HTML and/or PHP code can be included in another PHP script by using the **require** statement. If the file being included cannot be found, the script halts with a fatal error.

heading.php

```
<h3>Date: <?php $today = date("D M d Y"); echo $today; ?></h3>
```

The code above can be used in a PHP script like this (assuming it resides in the same directory):

```
<?php
    require 'heading.php'; // Prints Date: Fri Aug 29 2008
?>
```

B. The **include** statement does the same thing, but the script does not produce a fatal error if the included file is not found.

C. The **require\_once** and **include\_once** statements do the same thing as **require** and **include**, but they will not reload a file that has already been included.

## XIV. Web Input and Sessions

A. Form data can be accessed using the **superglobal arrays** `$_GET` and `$_POST`.

1. **\$\_GET**: for accessing data in the query string or URL (values are unescaped).

Query string: `custname=Bob+Smith&custage=21`

```
$name = $_GET["custname"];    // $name is "Bob Smith"
$age = $_GET["custage"];     // $age is 21
```

2. **\$\_POST**: for accessing posted for data from standard input (values are unescaped).

STDIN: `custname=Bob+Smith&custage=21`

```
$name = $_POST["custname"];   // $name is "Bob Smith"
$age = $_POST["custage"];     // $age is 21
```

3. Always a good idea to use **isset** to check if the variable exists in `$_POST` and `$_GET` before accessing to avoid PHP warnings:

```
if (!isset($_POST["custname"]) || $_POST["custname"] == "")
    echo "The customer's name was left blank.";
```

4. Shortcut to accessing variables in `$_GET` and `$_POST`: **extract** puts all key/value pairs in identically named variables.

```
extract($_POST);
if (isset($custname))
    echo "Hello, $custname!";
```

B. Sessions – To keep track of data between HTTP requests, data can be stored in cookies using the **\$\_COOKIE** array, or it can be stored on the web server via session variables in the **\$\_SESSION** array.

1. **\$\_COOKIE** – for accessing HTTP cookies (stored on the client).

```
// Set cookie which expires in 24 hrs
setcookie("name", $name, time() + 60 * 60 * 24);

// Set cookie which expires with the session
setcookie("age", $age);

echo $_COOKIE["name"];    // Print contents of name cookie
```

2. **\$\_SESSION** – for accessing session variables (stored on the web server, associated with the session ID).

```
// Must be first line in the script; creates a session if one doesn't already exist
session_start();

echo session_id();           // Prints the session ID
$_SESSION["myauto"] = "convertible"; // Set session var

$auto = $_SESSION["myauto"]; // Get session var
session_destory();          // Clears all session vars
```

C. Other superglobals:

1. **\$GLOBALS** - all variables which are currently defined in the global scope of the script.

2. `$_ENV` – for accessing data about the PHP parser's environment.
3. `$_FILES` – for HTTP file uploads.
4. `$_REQUEST` – contains all variables in `$_GET`, `$_POST`, and `$_COOKIE`
5. `$_SERVER` – for accessing web server environment variables like `REQUEST_METHOD`.

```
if ($_SERVER["REQUEST_METHOD"] == "GET")
```

## XV. Classes and Objects

- A. PHP supports many object-oriented programming concepts like constructors, destructors, abstract classes and methods, interfaces, dynamic creation of members and methods, etc. For a complete discussion, see <http://www.php.net/manual/en/language.oop5.php>.
- B. Declare a base class with a constructor and destructor. The `__toString` method is useful for serializing the object to a string.

```
class SuperPerson
{
    public $Name;           // Accessible to anyone
    public $PowerLevel;

    // Constructor
    public function __construct($Name, $PowerLevel = 0) {
        $this->Name = $Name;
        $this->PowerLevel = $PowerLevel;
    }

    // Destructor called when object goes out of scope
    function __destruct() {
        echo "Bye-bye";
    }

    // Convert object to string representation
    public function __toString() {
        return "Name = $this->Name, PowerLevel = $this->PowerLevel\n";
    }
}
```

- C. Extend the base class from above to create a super hero:

```
class SuperHero extends SuperPerson
{
    private $browniePoints = 0;           // Accessible only within the class

    public function Save($victim) {
        echo "$this->Name is saving $victim.\n";
        $browniePoints++;
    }
}
```

- D. Declaring and using objects:

```
$hero = new SuperHero("Spam-Man", 3);
echo "$hero";           // Prints Name = Spam-Man, PowerLevel = 3
$hero->Save("Laura Jones");
```

## XVI. Database Connectivity

- A. Most popular databases are supported including MySQL, Oracle, MS Access, SQL Server, etc.
- B. Many PHP developers use MySQL (<http://www.mysql.com/>) because of its cost (free in most cases) and durability.
- C. Full listing of MySQL functions are at <http://www.php.net/manual/en/ref.mysql.php>
- D. Connecting to MySQL (must be called once before executing any database operations):

```
mysql_connect($hostname, $username, $password);
```

- E. Select database to access (must be called after connecting and before executing any database operations):

```
mysql_select_db($database);
```

- F. Common database operations:

### 1. Query – Using the SELECT statement

```
$query = "SELECT ID, Name FROM Students WHERE GPA >= 2.0";
$result = mysql_query($query) or die(mysql_error() . "<br>query = $query<br>");
while ($row = mysql_fetch_row($result)) // Stops when no more rows are left
    echo "ID=$row[0] name=$row[1]<br>\n";

// Same thing using an associative array
while ($row = mysql_fetch_array($result))
    echo "ID=$row[ID] name=$row[Name]<br>\n";

// Test to see if no record is returned
$query = "SELECT Name FROM Students WHERE ID = 123";
$result = mysql_query($query) or die(mysql_error());
$row = mysql_fetch_array($result);
if (!$row)
    echo "Student not found. ";
else
    echo "Hello, $row[Name]! ";
```

### 2. Insert – Using the INSERT statement

```
$cmd = "INSERT INTO Students VALUES (789, 'Jane', 2.5)";
$result = mysql_query($cmd) or die(mysql_error());
$rows_inserted = mysql_affected_rows(); // Should return 1
echo "Successfully inserted $rows_inserted row.";
```

### 3. Update – Using the UPDATE statement

```
$cmd = "UPDATE Students SET GPA=3.1 WHERE ID = 123";
$result = mysql_query($cmd) or die(mysql_error());
$rows_updated = mysql_affected_rows(); // Should return 1
echo "Successfully updated $rows_updated row.";
```

#### 4. Delete – Using the DELETE statement

```
$cmd = "DELETE FROM Students WHERE GPA < 2.0";  
$result = mysql_query($cmd) or die(mysql_error());  
$rows_deleted = mysql_affected_rows();  
echo "Successfully deleted $rows_deleted row(s).";
```

- G. Protecting against SQL injection attacks is necessary to protect the integrity of the SQL statement. Always clean input data used in an SQL statement with the `mysql_real_escape_string` function.

```
$bad_name = $_POST["name"]; // "' OR ''='"  
$good_name = mysql_real_escape_string($bad_name); // Returns "' OR '\''='\'  
$query = "SELECT * FROM Students WHERE Name = '$good_name'";
```

© 2009 by Frank McCown